VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULITMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

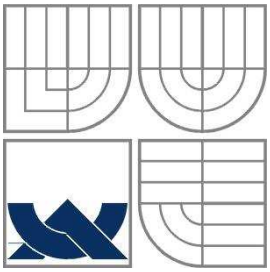
ZPRACOVÁNÍ ČEŠTINY V PYTHONU

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

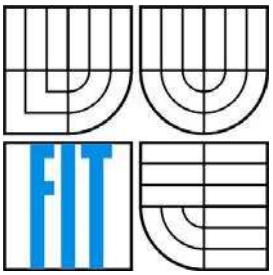
AUTOR PRÁCE
AUTHOR

ZDENĚK NOVOTNÝ

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMUTER GRAPHICS AND MULTIMEDIA

ZPRACOVÁNÍ ČEŠTINY V PYTHONU

PROCESSING CZECH IN PYTHON

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE — ZDENĚK NOVOTNÝ
AUTHOR

VEDOUCÍ PRÁCE — Doc. RNDr. PAVEL SMRŽ, Ph.D.
SUPERVISOR

BRNO 2009

Zadání bakalářské práce/6104/2008/xnovot85

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2008/2009

Zadání bakalářské práce

Řešitel: **Novotný Zdeněk**
Obor: Informační technologie
Téma: **Zpracování češtiny v Pythonu**
Kategorie: Web

Pokyny:

1. Seznamte se systémem NLTK.
2. Navrhněte a implementujte rozšíření systému, které umožní zpracování češtiny .
3. Srovnejte použitelnost pro češtinu s dalšími systémy.

Literatura:

- podle dohody

Při obhajobě semestrální části projektu je požadováno:

- funkční prototyp řešení

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Smrž Pavel, doc. RNDr., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2008
Datum odevzdání: 20. května 2009

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

V této práci jsou představeny některé způsoby zpracování českého jazyka. První část obsahuje obecný popis systému NLTK. Některé později zmíněné funkce byly inspirovány funkcemi ze systému NLTK. Jsou zde popsány funkce zabývající se časováním a skloňováním slov různých slovních druhů v českém jazyce. Další část je zaměřena na zpracování textu v českém jazyce, v němž dochází k vyhledávání a označení jednotlivých vět a jiných částí. Poslední část popisuje možnost aplikace transformačních pravidel na části textu. Výsledek po aplikaci pravidel lze zobrazit graficky.

Klíčová slova

zpracování přirozeného jazyka, Český jazyk, NLTK, NLP

Abstract

This bachorelor´s thesis presents some ways of Czech language processing. The first part contains a general destription of NLTK system. Some of aftermentioned functions were inspired by NLTK functions. There are described functions which attend to inflection and inflexion of various words class in Czech language. Next part is focused on processing of the text in Czech language in which are found and marked each sentences and other parts. Last part describes possibillity of tranformations rules application for each part of text. Results after rules application could be represented graphically.

Keywords

natural language processing, Czech language, NLTK, NLP

Citace

Zdeněk Novotný: Zpracování Češtiny v Pythonu, bakalářská práce, Brno, FIT VUT v Brně, 2009

Zpracování Češtiny v Pythonu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením docenta Pavla Smrže. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

…….
Zdeněk Novotný
20.května 2009

Poděkování

Chtěl bych využít této příležitosti, abych poděkoval svému vedoucímu práce, panu docentu Smržovi, za vedení a odbornou pomoc.

Obsah

Seznamy

Příklady

Obrázky

Tabulky

1 Úvod

Přirozený jazyk je denně užíván k běžné komunikaci mezi lidmi. Proto se brzy začala rozvíjet snaha o vytvoření nástrojů ke zpracování přirozeného jazyka.

Zpracování přirozeného jazyka je obor počítačové vědy zaměřený na interakci mezi počítačem a člověkem. Pro označení zpracování přirozeného jazyka budeme dále užívat zkatku NLP (z anglického *Natural Language Processing*).

Mezi příklady operací prováděných v rámci NLP patří automatický překlad(strojový překlad z jednoho jazyka do druhého), převod textu na mluvené slovo, převod mluveného slova na text, rozpoznávání entit v textu(vyhledávání vlastních jmen, slovních spojení atd.), strojové odpovídání na zadané otázky(program příjme otázku v přirozeném jazyce a odpoví na ní), extrakce informací z textu a mnoho dalších.

V průběhu let se všude ve světě, především na univerzitách, zakládaly projekty věnující se NLP. NLP není pouze záležitostí univerzit,ale své projekty rozvíjí i soukromý sektor. Mezi projekty patří například ***OpenPipeLine, openNLP, Gate, Uima, LingPipe, NLTK*** a mnoho dalších.

Kapitola následující za úvodem se věnuje obecnému popisu systému ***NLTK***[1]. Z dokumentace této knihovny byly čerpány některé informace pro tuto bakalářskou práci. V následující části jsou uvedeny jiné nástroje pro NLP a celkové hodnocení použitelnosti pro český jazyk.

Funkce obsažené ve skriptu *cip.py* se věnují morfologii českého jazyka. Jsou zde popsány operace skloňování a časování slovních druhů, lemmatizace. Dále zde nalezneme podpůrné funkce k jiným skriptům.

Obsah *split.py* se zaměřil na zpracování nestrukturovaného textu. Je zde popsán způsob označení jednotlivých vět a dalších části textu.

Popis skriptu *chunky.py* se věnuje získávání informací z textu a následnému zobrazení.

[1]Natural Language Toolkit, více v [3]

2 NLTK

Následující kapitola popisuje obecné vlastnosti nástroje NLTK. Infomace jsou čerpány z [11] a [3].

2.1 Základní informace

Počátek vzniku nástroje sahá na přelom minulého tisíciletí. V této době na Univesity of Pennsylvania, Philadelphia, pánové Edward Loper a Steven Bird spolu se svými tehdejšími studenty pokládají základní kameny tohoto systému. Původně byl NLTK vyvíjen jako základní nástroj pro studenty studující NLP. Celý tento nástroj je šířen pod GPL open source licencí. Více v [3].

Jako implementační jazyk byl vybrán jazyk Python. Tento jazyk nám umožňuje psát jednoduché strukturované programy (obsahuje transparentní syntaxi a sémantiku). Užitečná je i podpora grafické knihovny při vytváření uživatelských rozhraní. Díky použití jazyka Python je nástroj NLTK použitelný prakticky na všech platformách. Jediným požadavkem je podpora jazyka Python. Mezi dostupné platformy patří: Windows, OS X, Linux a Unix.
Obecné vlastnosti, které dělají z NLTK velice užitečný nástroj jsou:

- **Jednoduché užívání –** jednoduché ovládání. Hlavním úkolem je efektivní užití nástroje v praxi.
- **Konzistentnost** - nástroj musí mít konzistentní datovou strukturu a rozhraní.
- **Rozšířitelnost** – jednoduché přidávání nových komponent a jejich užití.
- **Dokumentace** - všechny struktury a jejich implementace jsou jednoduše a srozumitelně komentované.
- **Modulárnost** – spolupráce mezi různými komponentami v nástroji je omezena na minimum. Při komunikaci užívají jednoduchá rozhraní. Umožňuje nám to užívat některé komponenty nástroje bez větší závislosti na ostatních komponentách. Modulárnost souvisí s jednoduchým přidáváním nových komponent.

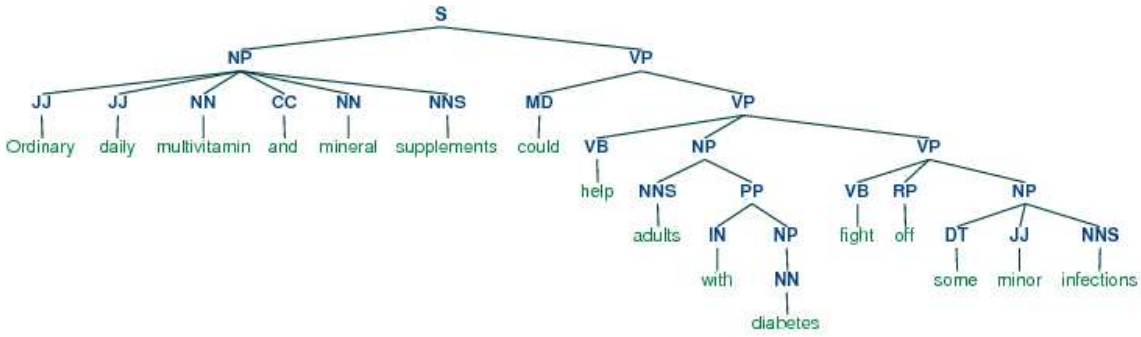
2.2 Moduly

Nástroj je implementován jako soubor nezávislých modulů. Každý modul se zabývá jinou operací v rámci NLP.

V současné době obsahuje tyto moduly(použiji originální nazvy v angličtině):

- **Corpus reader**(rozhraní pro čtení korpusů-zdroje statistických dat).
- **Tokenizers**(bílé znaky, slova, regulární výrazy).
- **Stemmers**(Porter, Lancaster).
- **Taggers**(anotování textu. Mezi užívané algoritmy patří: regexp, n-gram, backoff, Brill, HMM, TnT)
- **Chunkers**(jména entit)
- **Parsers** (reprezentace struktury textu)

V příkladě 2.1 vidíme ukázku reprezentace struktury pomocí modulu *draw.tree* pro vizualizaci výsledku.



Obrázek 2.1 – Vizualizace výsledku
(převzato z [3])

- **Semantic interpretation**(lambda calculus, DRT)
- **WordNet**(rozhraní ke slovníku WordNet, ostatní slovníky)
- **Classifiers**
- **Clusterers**
- **Metrics**
- **Estimation**
- **Miscellaneous**
- **NLTK-Contrib**

2.3 Dokumentace

Jak jsem již dříve zmínil, tak NLTK obsahuje podrobnou dokumentaci popisující užití nástroje a jeho strukturu. Celá dokumentace se dělí na tři základní kategorie:

Tutoriály
Tutoriál se zaměřuje na popis jednotlivých komponent. K popisu principu a funkčnosti komponenty se využívají příklady, na kterých se ukazuje chování komponenty. Všechny příklady jsou doplněny podrobnými komentáři.

Referenční Dokumentace
Obsahuje podrobný popis každého mudolu, rozhraní, třídy, metody, funkce a proměných obsažených v nástroji NLTK.

Technické zprávy
Vysvětlují design a implementaci nástroje. Jsou využívány vývojáři k dokumentování konstrukce nástroje.

2.4 Ukázka použití

V ukázkovém příkladu si představíme použití modelu *chunk* při tzv. chunkování, což je způsob získávání informací z textu. Ukázka je převzata z [3]. Na příkladě 2.1 vidíme příkazy, pomocí kterých přilinkujeme modul *chunk* a další potřebné funkce.

```
>>> from nltk.chunk import *
>>> from nltk.chunk.util import *
>>> from nltk.chunk.regexp import *
>>> from nltk import Tree
```

Příklad 2.1 – Přilinkování modulu chunk

Dalším krokem je vložit testovací data. To vidíme na příkladě 2.2.

```
>>> tagged_text = "[ The/DT cat/NN ] sat/VBD on/IN [ the/DT mat/NN ]
                  [ the/DT dog/NN ] chewed/VBD ./."
>>> gold_chunked_text = tagstr2tree(tagged_text)
>>> unchunked_text = gold_chunked_text.flatten()
```

Příklad 2.2 - Testovací data

Proměnná *tagged_text* z příkladu 2.2 obsahuje větu, kde každé slovo má přiřazený tagg(značku). Dále obsahuje skupiny slov, které jsou ohraničeny hranatými závorkami. Tyto skupiny se nazývají *chunk*. Tento prvek obsahuje určitou podskupinu slov ve větě. Podskupinou je myšleno slovní spojení nebo samostané slovo.

V proměnné *unchucked_text* jsme dostali upravenou větu, která obsahuje jednotlivá slova a jejich tagg, ale už neobsahuje hranaté závorky.

Nyní je potřeba zadat pravidla, která definují způsob shlukování slov do *chunku.* V příkladě 2.3 máme ukázku základních pravidel užívaných v modulu *chunk*.

```
>>> chunk_rule = ChunkRule("<.*>+", "Chunk everything")
>>> chink_rule = ChinkRule("<VBD|IN|\.>",
                           "Chink on verbs/prepositions")
>>> split_rule = SplitRule("<DT><NN>", "<DT><NN>",
                           "Split successive determiner/noun pairs")
```

Příklad 2.3 - Pravidla

Jendotlivá pravidla obsahují výraz zastupující pravidlo a textový popis funkce pravidla.
Každý výraz zastupující pravidlo se musí převést na regulární výraz. Například pravidlo *chunk_rule* spojí všechny slova ve větě do jednoho *chunku.*

```
>>> chunk_parser = RegexpChunkParser([chunk_rule], chunk_node='NP')
>>> chunked_text = chunk_parser.parse(unchunked_text)
>>> print chunked_text
(S
  (NP
    The/DT
    cat/NN
    sat/VBD
    on/IN
    the/DT
    mat/NN
    the/DT
    dog/NN
    chewed/VBD
    ./.))
```

Příklad 2.4 – Výsledek operace chunk

Proměnná *chunk_parser* obsahuje pravidlo převedené na regulární výraz a nové pojmenování pro nově vzniklí *chunk*. V proměnné *chunked_text* dostaneme výsledek po aplikování zadaného pravidla.
A posledné příkaz v příkladu 2.4 zobrazuje strukturu výsledné věty.

2.5 Zhodnocení

V současné době NLTK obsahuje velký soubor korpusů, rozsáhlou dokumentaci a stovky cvičných příkladů. To dělá z NLTK unikátní a komplexní systém určený pro studenty a ostatní vývojáře, kteří se zabývají NLP. Momentálně nástroj obsahuje 100 000 řádků kódu v jazyce Python. Podporuje operace: práce s korpusy, chunking, parsování, jazyková modelace, sémantická modelace, clustrování a mnoho dalších operací. Díky svému rozsahu a komplexnosti je NLTK užíváno na více než 60 univerzitách ve 20 zemích světa.

2.6 Instalace

Prvním krokem, před samotnou instalací NLTK, je kontrola zda máte ve svém počítači naistalovaný jazyk Python. Pokud nemáte tento jazyk na svém počítači naistalovaný, můžete použít odkaz[2], kde si stáhnete instalační soubor.

Po úspěšné instalaci jazyka Python můžeme přistoupit k instalaci NLTK. Na stránkách nástroje[3] si stáhneme balíčky dat pro Váš typ operačního systému.

2.6.1 Windows

Všechny stažené balíčky dat rozbalíme. Všechny získaná data umístíme do složky *C:\nltk_data*. Poté je nutné vytvořit uživatelskou proměnou prostředí. Postup je následující: klikneme na Tento počítač, klikneme na pravé tlačítko myši a vybereme vlastnosti. Ve vlastnostech si vybereme záložku *upřesnit*. U spodní části okna klikneme na tlačítko *proměnné prostředí* a pod oknem uživatelské proměnné klikneme na tlačítko *nová*. Název proměnné bude *NLTK_DATA* a hodnota bude obsahovat cestu *C:\nltk_data*;. Poté všechno potvrdíme tlačítkem OK. Instalace je hotová. Nyní můžeme plně využívat nástroj NLTK.

2.6.2 Linux/Unix

Instalace v prostředí Linux/Unix je také velice jednoduchá. Po stažení balíčku NLTK jej rozbalíme. Vznikce nám nová složka např. *nltk-0.9.9.* Otevřeme terminál a pomocí příkazu *cd* se dostaneme do dané složky. Nyní stačí pouze spustit příkaz:

```
sudo python setup.py install
```

Instalace je dokončena.

[2] http://www.python.org/ftp/python/2.5.4/python-2.5.4.msi

3 Ostatní obecné nástroje pro NLP

3.1 OpenNLP

Tento projekt obsahuje soubor jednotlivých nástrojů pro NLP. Snahou projektu ***OpenNLP*** je sdružovat tzv. „*open source*“[3] projekty zabývající se NLP. Hlavním úkolem je tedy umožňovat a podporovat spolupráci mezi vývojáři jednotlivých projektů. Primárním jazykem pro projekty a nástroje je Java.

Soubor nástrojů je primárně určen k integraci do jiných systémů zpracovávajících text. Jednotlivé balíčky s nástroji nabízí standardní škálu funkcí pro práci s textem. Například hledání slov a části textu, vyhledávání a rozpoznávání jednotlivých vět, statistické vyhodnocení slov,anotování slov a mnoho dalších funkcí. Podpora jednotlivých jazyků je závislá na existenci modelu. Pod pojmem model rozumíme soubor upravených obecných funkcí pracujících s daným jazykem a zdroj informací pro jazyk. Momentálně jsou k dispozici modely pro Anglický, Španělský, Německý a Thajský jazyk. Tyto modely jsou dostupné v nové verzi distribuce.

Snahou projektu ***OpenNLP*** je stát se jedním z hlavních zdrojů pro nástroje zabývající se NLP. V současné době se tedy snaží o vývoj projektu ***Maxent***. Náplní tohoto projektu je vytvořit maximální počet modelů pro různé jazyky. Spolu s tímto projektem jsou vyvíjeny další projekty založené na ***Maxent***. Pokud se úspěšně podaří dokončit zmíněné projekty, tak se může ***OpenNLP*** stát jedním z hlavních univerzálních nástrojů pro NLP. Pro samotné studium funkcí nástroje je dostupná dokumentace [10].

Využití pro Český jazyk zde není momentálně reálné, vzhledem k neexistenci modelu. Po dokončení projektu ***Maxent*** se uvidí, zda se bude moci tento soubor nástrojů užít pro Český jazyk. Další možností je upravit si zdrojové kódy s ohledem na *open source* licenci a pokusit se ho uzpůsobit sami. Informace čerpány z [10].

3.2 LingPipe

Nástroj je vyvíjen společností Alias-i. Distribuce tohoto nástroje závisí na způsobu jeho využití. K dispozici je několik variant licencí. Celý nástroj je naprogramován v jazyce Java a nabízí velkou škálu funkcí pro práci s přirozeným jazykem. Standardem jsou funkce pro značkování textu, vyhledávání slov či částí textu, dělení slov, spolupráce se slovníky, užíváni dokumentů z internetu, překlady textu atd.

Primárně podporovanými jazyky jsou zde Angličtina a Čínština, ke kterým jsou dostupné modely. Avšak je zde snaha užívat tento nástroj v globálním měřítku. Všechny texty jsou převáděny do jednotného unicode kódování. Následně jsou texty zpracovány. Jedna z dalších výhod je podpora různých formátů uložení textu, který se zpracovává. Mezi podporované formáty patří XML, HTML či volný text.

Pro studium samotného nástroje je dostupná dokumentace, tutoriály a ukázková dema. Vše je dostupné na stránkách nástroje [9]. Použitelnost pro český jazyk je možná.

[3] Počítačový software s otevřeným kódem. Za určitých podmínek může uživatel zdrojový kód prohlížet a upravovat.

3.3 Celkové zhodnocení

Pokud bych celkově hodnotil obecné nástroje pro NLP a jejich využití pro český jazyk, došel jsem k několika omezením. Je nutné otestovat přesnost výsledků při užití daných nástrojů a mít k dispozici dostatečný zdroj českých dat. Při užití nástrojů v interakci s jazykem Python je nutné vytvořit rozhraní mezi oběma jazyky.

4 Implementace

4.1 Skript cip.py

Funkce v tomto rozhraní se převážně zabývají morfologií. Morfologie, čili tvarosloví lze rozdělit na skládání a odvozování. První část se zabývá skloňováním slovních druhů mezi než patří podstatná jména, přídavná jména, zájmena a číslovky. V druhé části dochází k časování slovních druhů. Mezi slovní druh, který lze časovat, patří slovesa. Všechny zmíněné slovní druhy patří do kategorie ohebných slovních druhů.

Dalšími užitečnými funkcemi, které zde nalezneme, je funkce vyhledávající tvary a odvozeniny zadaného slova či kontrola, zda zadané slovo patří mezi zkratky. Všechny tyto funkce používají morfologický analyzátor a slovník kolegy Bc. Stanislava Černého [5]

4.1.1 Skloňování slovních druhů

Skloňování, tedy ohýbání slovních druhů, nám umožňuje vyjádřit různé mluvnické kategorie. V závislosti na rozsahu informací, které obsahuje mnou užívaný slovník, jsem si vybral pro práci mluvnické kategorie *číslo* a *pád*. Tyto mluvnické kategorie jsou společné pro všechny slovní druhy u nichž lze skloňování provádět a lze skrze ně lehce rozlišovat tvary slov. Český jazyk rozlišuje 7.pádů a dvojí mluvnické číslo: *jednotné*(singular) a *množné*(plural)[1].

Operací skloňování se v tomto rozhraní zabývá funkce ***inflect***. Parametry, které tato funkce přijímá jsou, *word* – slovo, jež se má skloňovat, *number* - určuje číslo a *case* – pro pád. První krok v transformaci je kontrola, zda dané slovo je v používaném slovníku. Pokud se dané slovo ve slovníku nevyskytuje, je vráceno v nezměněné podobě zpět. Samozřejmostí je kontrola zadaných parametrů, jestli jsou správně zadány. Akceptovatelné parametry u *number* jsou: *„S“* pro jednotné číslo a *„P“* pro množné číslo. Parametr case přijímá čísla 1-7 značící příslušný pád. Po předešlých operacích dochází k vyhledání všech morfologických kategorií pro dané slovo. Touto operací jsme získali slovní druh a základní slovo, od kterého je naše slovo odvozeno. Ze všech možných nalezených variant jsou vyřazeny základní slova, u kterých slovní druh nepovoluje operaci skloňování. Pro každé nalezené základní slovo najdeme všechny tvary. Nyní přistupujeme k poslední operaci. U každého slova ze souboru všech variant je určena anotace, čili informace o daném tvaru slova. Každá z těchto anotací se pozmění dle zadaných parametrů. Na příkladu 4.1 si ukážeme, jak obecně může vypadat anotace a které části nás zajímají.

anotace pro(*„okno“*): k1gNnSc5

k1 - značí slovní druh
(zde je to podstatné jméno)
nS – mluvnické číslo
(zde je to S-singular)
c5 - pád
(zde je to 5.pád)

Příklad 4.1 – Anotace pro podstatné jméno

Poté, co změníme v anotaci parametry pro mluvnické číslo a pád na základě zadaných parametrů, dochází k vyhledávání tvaru slova určeného anotací. Daný tvar se odvozuje ze základního slova. Po vyhledávání pro všechna základní slova a anotace se soubor jedinečných výsledků navrátí zpět.

4.1.2 Časování slovních druhů

Časování sloves nám pomáhá vyjádřit osobu, číslo, čas, rod, vid a způsob, jenž souvisí s dějem či vztahem, které dané sloveso popisuje. V mém případě pro popis slovesa a jeho tvaru jsem vybral mluvnické kategorie *osoba*, *číslo* a *čas*. Vybrané kategorie jsem zvolil pro jejich jednoduché určení pro dané slovo. U zbývajících kategorií by bylo pro uživatele obtížné je správně určit. Mluvnická kategorie *osoba* rozeznává tři základní typy: 1.osoba(já, my), 2.osoba(ty,vy) a 3.osoba(on, ona, ono, oni, ony, ona) [2]. Kategorie číslo rozeznává jednotné a množné číslo. A poslední sledovaná kategorie rozeznává přítomný, budoucí a minulý čas.

Průběh zpracování informací ve funkci ***inflex,*** která provádí operaci časování, je podobný jako u předchozí funkce. Parametry pro tuto funkci jsou *word*-slovo, *person*-osoba, *number*-číslo a *time*-čas. Zkontroluje se, zda dané slovo je v užívaném slovníku. Pokud dané slovo není ve slovníku, je vráceno zpět beze změny. Provede se kontrola správnosti zadaných parametrů. Vyhledáme všechna základní slova, která odpovídají slovnímu druhu, u něhož lze provést skloňování. Nyní se vyhledají všechny varianty pro nalezená základní slova. U každé nalezené varianty dochází ke změně anotace. V příkladě 4.2 vidíme, které části se mění:

anotace pro(*„utíkat“*) k5eAaImIp2nS

k5 – slovní druh
(zde je to sloveso)
mI – mód(určuje čas)
(zde přit. čas)
p2 - osoba
(zde 2.osoba)
nS - mluvnické číslo
(zde S-singular)

Příklad 4.2 – Anotace pro sloveso

4.1.3 Lemmatizace

Princip lemmantizace je vyhledání všech variant a tvarů pro zadané slovo. Touto operací se zabývá funkce ***lemma***. Parametry pro tuto funkci jsou *word*-dané slovo a *number*-určuje počet slov, která se mají vytisknout na jeden řádek.

Tato operace provádí nalezení základního slova, od kterého bylo odvozené zadané slovo. Poté se naleznou všechny tvary a varianty odvozené od základních slov. Celá výsledná množina se všemi nalezenými řešeními je srovnána tak, aby se zde žádná slova neopakovala. V posledním kroku se výsledek vypíše na standardní výstup. Parametr *number* určuje počet z nalezených řešení, který se vypíše na jeden řádek.

4.1.4 Skloňování slovních spojení

Skloňování slovních spojení zajišťuje funkce ***inflection.*** Parametry funkce jsou *words*-slovní spojení, *case*-pád. Dochází k dělení věty na jednotlivá slova a jejich následnému zpracování. Jako výsledek je navrácen soubor všech možných kombinací z nalezených tvarů.

4.1.5 Časování slovních spojení

Pokud je třeba provést časování v slovním spojení, lze využít funkci ***inflexion.*** Přijímané parametry jsou *words*-slovní spojení, které se má zpracovat. Tato funkce využívá pro časování dříve zmíněné funkce. Prvním krokem, který se provede, je dělení zadané věty na jednotlivá slova a interpunkční znaménka. Následně jsou všechny prvky věty zpracovány samostatně. Ve výsledku vrací funkce soubor spojení obsahující kombinace všech nalezených tvarů a variant.

4.1.6 Ostatní funkce

Funkce ***short*** je využívána jako podpůrná funkce pro skript *split.py*. Jediným přijímaným parametrem je kontrolované slovo. U zadaného slova se naleznou všechny jeho možné slovní druhy a kategorie. Pokud slovo spadá do kategorie zkratek, je vrácen příznak určující příslušnost daného slova ke zkratkám.

Úkolem funkce ***WClass*** je nalézt všechny slovní druhy, kterých může zadané slovo nabývat. Množina všech slovních druhů je vrácena jako výsledek funkce.

4.2 Skript split.py

Velice často se stává, že je nutné zpracovat nestrukturovaný text do podoby, kterou lze dále upravovat. Minimálním požadavkem je vyznačení jednotlivých vět v zadaném textu. Dalšími požadavky může být vyznačit přímé řeči a citáty v jednotlivých větách.

Za nestrukturovaný text lze považovat takový, kde nejsou vyznačené jednotlivé věty, text může být napsán na jednom „nekonečném“ řádku, či každé jednotlivé slovo je napsáno na jednom řádku.

Hlavní funkce, která obstarává zpracování textu je funkce ***Split***. Přijímá parametry *file* – název souboru nebo vstupní řetězec, *input* – příznak určující zda vstupem je soubor či řetězec.

4.2.1 Odlišení vstupního a výstupního textu

Pokud je zadáno, že vstupní text se načítá ze souboru, je třeba vytvořit nový soubor, do kterého se bude ukládat výsledek. Nový soubor je vytvořen ve stejné složce, kde je umístěn původní soubor. Název nového souboru je složen ve funkci ***Name,*** která k původnímu názvu přidá zkratku *„_split“*, odlišující nový soubor od starého. Nový název souboru je *název_split[.přípona.]*. Pokud je vstupem textový řetězec, tak se nový soubor nevytváří a výsledek bude vrácen opět ve formě textového řetězce.

4.2.2 Druhy značek pro odlišení části textu

Před samotným popisem způsobu vyhledávání a značkování dílčích částí textu je třeba ukázat jakými značkami jsou dílčí části označeny(viz. Příklad 4.3).

<S>….</S> - tyto značky označují každou samostatnou větu.

<SD>….</SD> - pomocí těchto značek značíme přímou řeč,citát apod.

Příklad 4.3 –Rozlišovací značky v textu

4.2.3 Přímá řeč, citáty atd.

V českém jazyce jsou přímé řeči, citáty atd. uvozeny pomocí uvozovek. Rozlišujeme několik základních druhů uvozovek [6]. Na příkladu 4.4 vidíme uvozovky užívané v českém jazyce a uvozovky, které detekujeme v naší funkci.

„“ - užívané v českém jazyce.

"" - užívané především v anglickém jazyce.

»« - užívané v českém jazyce.

Příklad 4.4 – Druhy uvozovek

Označené přímé řeči a citáty jsou považovány za zpracované a dále se neupravují.

4.2.4 Jednotlivé věty

Český jazyk užívá pro ukončení věty interpunkční znaménko tečku, vykřičník a otazník. Nejproblematičtější je rozhodování o významu znaménka tečky ve větě. V tabulce 4.1 vidíme ukázky výskytu tečky a způsob zpracování.

Výskyt tečky	Význam tečky	Prováděná operace
číslice.číslice (př. 13.00)	desetinna tečka v čísle	**NEDĚLÍME**
zkratka.slovo (př. Mgr.Hrubeš)	tečka za zkratkou.	**NEDĚLÍME**
slovo.slovo (př.škola. Na)	tečka ukončuje větu.	**DĚLÍME**
…	vsuvka, nahrazují část textu.	**NEDĚLÍME**
číslice.měsíc (př. 13.září)	Datum	**NEDĚLÍME**

Tabulka 4.1 – Výskyt znaménka tečky ve větě

Je tedy velice důležité zpracovat každou část textu, kde se vyskytuje znaménko tečky, zvlášť. Všechny nalezené části jsou postupně zpracovány funkcí ***doSplit,*** která separuje část textu „před“ tečkou a „za“ tečkou. Následně je testováno zda nalezená varianta nespadá do jedné z možností v tabulce 4.1. Každé separované slovo nacházející se „před“ tečkou je testováno pomocí funkce short zda není ve slovníku uloženo v kategorii zkratky. Pokud se jedná o zkratku, dle tečky se text nedělí.

Výkonnost skriptu split.py můžeme zhodnotit dle naměřených hodnot (viz. tabulka 4.2)

Velikost souboru [MB]	Celkový čas zpracování [ms]
50	66480
100	132673
150	199253

Tabulka 4.2 – Rychlost skriptu split.py

Skript byl testován na stroji s konfigurací: FreeBSD 7.x, C2D 3 Ghz, 8GB RAM. Průměrný čas nutný ke zpracování jednoho MB dat je 1328,23 ms.

4.2.5 Příprava strukturovaného textu

V případech, kdy je třeba dříve označkovaný text ze souboru či textového řetězce rozdělit na jednotlivé věty a smazat dělící značky můžeme užít funkci ***PoSplit***. Tato funkce příjme na vstupu text ze souboru či řetězce a rozdělí ho dle obsažených značek na dílčí věty. Všechny značky jsou zde smazány. Výsledná množina jednotlivých vět je navrácena zpět.

4.3 Skript chunky.py

V předchozích kapitolách jsem popisoval způsoby zpracování textu zapsaného v českém jazyce. Mohli jsme v každém textu vyznačit jednotlivé věty, přímé řeči. Zde se dostáváme do fáze, kdy potřebujeme vyhledat v označeném textu určité části tak, abychom si je mohli zobrazit a dát do případných souvislostí.

Celý skript byl inspirován částí knihovny z **NLTK**[4]. Princip fungování skriptu se pokusím vysvětlit na jednoduchém příkladu. Díky praktické ukázce vidíme, co se děje v průběhu zpracování zadaného textu. Jako ukázkový příklad 4.4 použiji jednoduchou větu, kterou za pomocí zadaných pravidel transformuji. Transformace v tomto případě znamená nalezení částí textu, které jsou následně v textu nově označeny. Každé slovo ve větě (viz. příklad 4.4) má k sobě přiřazenou značku označující slovní druh .

Originální věta:

Sportem ku zdraví a trvalé invaliditě!

Označkovaná věta:

Sportem/POJ ku/PRE zdraví/POJ a/SPO trvalé/PRJ invaliditě/POJ !/note

Příklad 4.5 – Ukázková věta

Hlavní transformace probíhá pomocí funkce ***Chunk.*** Parametry, v této funkci přijímané, jsou *sentence* – označkovaná věta, která se má zpracovat; *rules* – soubor pravidel, jenž se mají na větu aplikovat, *flagy* – příznak určující způsob zobrazení výsledku.

4.3.1 Úprava věty pro vyhledávání

Pro zjednodušení práce s textem jsem se rozhodl zavést v textu *substituci*[5]. Převod zadané věty provádí funkce ***PreChunk***. Tato funkce v prvním kroku rozdělí danou větu na jednotlivé prvky.
K substituci se užívá značka přiřazená ke každému prvku ve větě. Ke každému prvku je tedy provedena transformace, kterou vidíme na příkladu 4.5.

Sportem/POJ => [‘Sportem/POJ’,’POJ’]

Příklad 4.6 – Transformace na substituční název

Souběžně s touto transformací se vytváří množina o dvou prvcích. Prvním prvkem množiny je soubor obsahující chronologickou posloupnost všech prvků ve větě. Druhý prvek je typu string, který

[4] Natural Language Toolkit, více v [3]
[5] Substituce – Nahrazení složitějšího výrazu čí části jednodušším výrazem

obsahuje substituce jednotlivých prvků věty. Substituce obsahuje číselnou pozici prvku věty, spolu se značkou daného prvku. Výsledek pro moji ukázkovou větu vidíme na příkladu 4.6.

[['Sportem/POJ', 'ku/PRE', 'zdraví/POJ', 'a/SPO', 'trvalé/PRJ', 'invaliditě/POJ'], '0/POJ 1/PRE 2/POJ 3/SPO 4/PRJ 5/POJ ']

Příklad 4.7 – Připravená věta k transformaci

Nyní jednotlivá slova ve větě zastupuje substituční označení. Při následném vyhledávání se odkazujeme pouze na dané označení.

4.3.2 Transformační pravidla

Před dalším popisem jednotlivých funkcí je třeba popsat druhy transformačních pravidel, která jsou užívána k transformaci věty. Náplní pravidla je určit, která část textu se má označit novou značkou. V příkladě 4.7 vidíme tři základní typy povolených pravidel.

{regexp} = chunk pravidlo
}regextp{ = chink pravidlo
regexp}{regexp = split pravidlo

Příklad 4.8 – Obecný vzor pravidel

Z obecného hlediska obsahuje každé pravidlo značku „*regexp*“, která je označením pro *regulární výraz*[6]. Jednotlivá pravidla jsou odlišená umístěním spojitých závorek.

Pravidlo **Chunk** funguje tak, že ve větě se naleznou všechny části odpovídající regulárnímu výrazu. Poté jsou všechny nalezené výsledky jednoduše v dané větě uzavřeny do spojitých závorek. Ukázku možného vzhledu pravidla vidíme v příkladě 4.8.

STE: {<POJ><PRE>}

Příklad 4.9 – Ukázka pravidla typu Chunk

U pravidla **Chink** dochází k malé změně ohledně zpracování nalezených výsledků. Opět se naleznou všechny části odpovídající regulárnímu výrazu, ale v tomto případě se uzavřou do spojitých závorek všechny ostatní části věty, které nejsou mezi nalezenými výsledky. Možný vzhled pravidla ukazuje příklad 4.9.

[6] Regulární výraz - Je řetězec popisující celou množinu řetězců. Nejčastější užíván ve skriptovacích jazycích k vyhledání a úpravě textu

STE: }<POJ><PRE>{

Příklad 4.10 – Ukázka pravidla typu Chink

Poslední v pořadí je **Split** pravidlo. Spojité závorky umístěné uprostřed pravidla nám oddělují dvě nová pravidla. Obě nově vzniklá pravidla jsou typu *Chunk.* Ukázka pravidla v příkladě 4.10.

STE: <CIT><SLO>*}{<POJ>

Příklad 4.11 – Ukázka pravidla typu Split

Nově vzniklá pravidla mají podobu shrnutou v příkladu 4.11.

STE: {<CIT><SLO>*} a **STE: {<POJ>}**

Příklad 4.12 – Dvě nová pravidla z pravidla typu Split

Podrobnější popis toho, co jednotlivé části pravidel znamenají a jak je vytvářet, najdeme v souboru *rules.pdf.* Tento soubor je přiložen v elektronické podobě.

4.3.3 Kontrola transformačních pravidel

Nyní se můžeme vrátit k popisu funkcí v tomto skriptu. Po popisu typu pravidel je třeba zkontrolovat jejich správný zápis. Základní kontrola pravidel je prováděna funkcí **CheckRule**. Soubor pravidel je zadán skrze parametr ***rules*** ve funkci *Chunk*. Tvar zadaných pravidel můžeme vidět na příkladu 4.12, kde je zadáno pravidlo pro moji ukázkovou větu.

['STE: {<POJ><PRE>}']

Příklad 4.13 – Ukázka souborů pravidel

U každého pravidla se kontroluje několik atributů. Na prvním ukázkovém pravidle v souboru si ukážeme které. Každé pravidlo se rozděluje na levou a pravou část, jak vidíte na příkladu 4.13.

STE - levá část.

: - oddělující znaky(dvojtečka a mezera).

{<POJ><PRE>} - pravá část .

Příklad 4.14 – Části pravidla

První kontrola zjišťuje, zda pravidlo obsahuje levou i pravou část. Každé pravidlo může obsahovat pouze jeden pár složených závorek, který svým umístěním rozlišuje jednotlivá pravidla, a proto se kontroluje správný výskyt složených závorek v pravé části pravidla. Poslední kontrola se provádí na jednotlivých prvcích pravé části. Na příkladu 4.14 vidíme správný zápis dílčího prvku.

<POJ> -samostatný prvek v pravidle.

Příklad 4.15 – Prvek pravidla

Každý prvek pravidla musí být uzavřen do páru hranatých závorek. Pokud pravidlo obsahuje všechny dříve zmíněné atributy, je vyhodnoceno jako vyhovující a dále se zpracovává.

4.3.4 Aplikace transformačních pravidel

Po přípravě zadané věty k transformaci a kontrole pravidel můžeme začít aplikovat jednotlivá pravidla. Pravidla jsou aplikována v pořadí, v jakém byla zadána v souboru pravidel.

Nejdříve se provede zjištění typu pravidla, které se má aplikovat. Pokud je dané pravidlo typu *Chunk*, je použita funkce ***ReChunk***, která dané pravidlo aplikuje.

Uvnitř funkce se oddělí levá část pravidla, která nám nově pojmenuje nalezené části. Pravá část pravidla je pomocí funkce ***Reg*** převedena na regulární výraz. Po těchto operacích dochází k vyhledání všech odpovídajících částí textu za pomoci regulárního výrazu. Vyhledávání se provádí ve větě zmíněné v příkladu 4.6, která obsahuje substituce jednotlivých prvků věty.

Všechny nalezené výsledky jsou opět uloženy do souboru částí vět a slov. Nové substituční označení pro danou část věty má opět podobu číselné pozice prvku v souboru a značky získané z levé části pravidla.

Ostatní typy pravidel jsou zpracovány podobným způsobem. Pro pravidlo typu *Chink* je tu funkce ***ReChink*** a pravidlo *Split* funkce ***ReSplit***. Kroky zpracování pravidla jsou získání nového pojmenování, převod na regulární výraz, vyhledávání, substituce nalezených výsledků. U typu *Split* dochází k rozdělení pravidla na dvě nová pravidla a následné zpracování každého pravidla zvlášť. Výsledek po aplikaci všech pravidel na ukázkovou větu můžeme vidět v příkladě 4.15.

[['Sportem/POJ', 'ku/PRE', 'zdraví/POJ', 'a/SPO', 'trvalé/PRJ', 'invaliditě/POJ'], '(STE 0/POJ 1/PRE)', '(KLI 4/PRJ 5/POJ)', '(PRC 3/SPO 7/KLI)'], '6/STE 2/POJ 8/PRC ']

Příklad 4.16 – Transformovaná věta dle pravidel

4.3.5 Zobrazení výsledku transformace

Výsledky transformace je nutné zobrazit uživateli. Způsob zobrazení je určen parametrem *flagy* ve funkci *Chunk*. Na výběr máme ze dvou možných variant. První varianta zobrazí výsledek na standardní výstup. U druhé varianty je výsledek zobrazen v grafické podobě.

Pro zobrazení použijeme funkci ***Print***. Předtím, než dojde k samotnému zobrazení je nutné provést úpravu výsledku. Pro obě varianty je nutné vrácený výsledek uvést do podoby, kde budou obsažena všechna slova a znaky z původní zadané věty. Ve výsledku zmíněném v příkladu 4.15 se prochází první část množiny, obsahující jednotlivé prvky věty, od posledního prvku k prvnímu. U každého prvku je zjištěna jeho číselná pozice. Nyní procházíme druhý prvek množiny zmíněný v příkladu 4.15, kde vyhledáme substituční označení obsahující číselnou pozici prvku. Následně je označení nahrazeno prvkem na dané pozici. Po ukončení substituce dostaneme množinu obsahující všechny části původní věty a všechny úpravy dle zadaných pravidel.
Na příkladě 4.16 vidíme výsledný soubor pro moji ukázkovou větu.

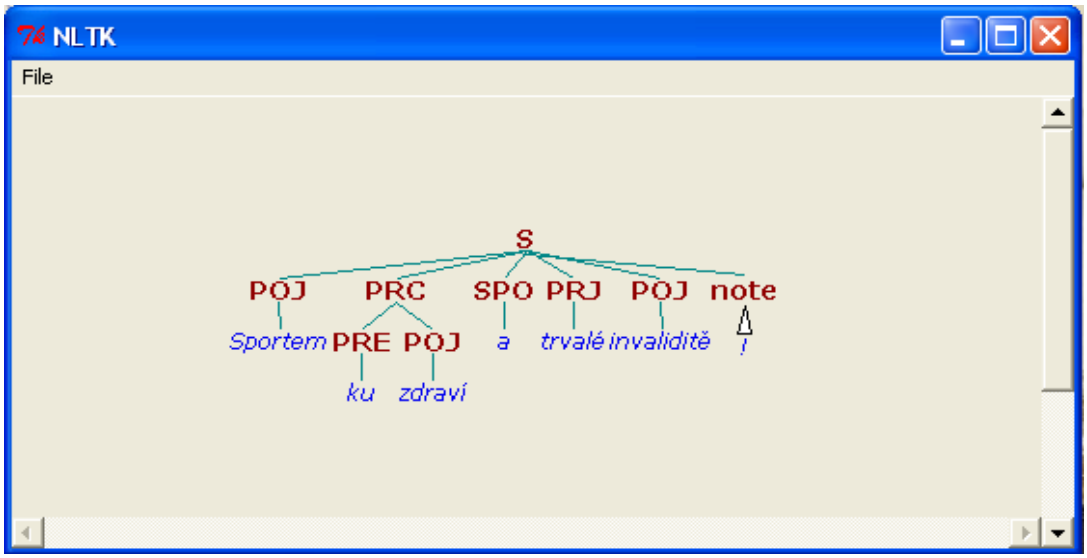
['(STE Sportem/POJ ku/PRE)', 'zdraví/POJ', '(PRC a/SPO (KLI trvalé/PRJ invaliditě/POJ))']

Příklad 4.17 – Soubor výsledných částí transformované věty

Po této úpravě je výsledek možný zobrazit na standardní nebo grafický výstup. U grafického zobrazení se využívá knihovna z NLTK [3]. Tato knihovna užívá kulaté závorky k odlišení jednotlivých prvků pro zobrazení. Bylo tedy nutné nahradit všechny kulaté závorky, které mají spojitost s významem původní věty, hranatými závorkami. Těmito operacemi byl výsledek připraven k zobrazení.
Na obrázku 4.1 vidíme zobrazení výsledku transformace ukázkové věty ve standardní výstup.

Obrázek 4.1 – Zobrazení ve standardní konzoly

A na obrázku 4.2 máme výsledek zobrazen graficky.



Obrázek 4.2 – Zobrazení v grafické podobě

Pokud v grafickém zobrazení kliknete na slovní druh, dochází k tzv. „sbalení“ slova. Při této operaci se jednotlivá slova sbalí k hierarchicky nadřazenému označení ve větě.

6 Závěr

Výsledkem mojí práce je soubor několika funkcí, které se dají použít při práci s českým jazykem. V této chvíli není tento soubor použitelný pro praxi. Spíše byla snaha vytvořit soubor základních funkcí, které by se postupně doplňovaly dalšími funkcemi. Byl využíván morfologický analyzátor a slovník kolegy Bc. Černého[5].

Soubor obsahuje funkce, které se věnují morfologii českého jazyka. Pomocí funkcí můžeme skloňovat či časovat slovní druhy. Operaci skloňování a časování lze provádět i na slovních spojeních.

Další část se věnuje zpracování nestrukturovaného textu. Za zdroje textu lze použít soubor či přímo textový řetězec. Vstupní text je zpracován tak, že je rozdělen na jednotlivé věty a ty jsou označeny značkami. Pokud nestačí základní vyznačení vět, je zde možnost vyznačit přímé řeči, citáty. Po označkování nestrukturovaného textu je k dispozici funkce, která označený text zpracuje a vrací soubor obsahující jednotlivé věty.

K dispozici je zde i funkce užitečná při extrakci dat z textu. V této funkci můžeme definovat určitá pravidla, která předepisují souvislosti mezi jednotlivými slovy. Následně jsou tato pravidla aplikována na označenou větu. Výsledek transformace lze zobrazit na standardní nebo grafický výstup.

Všechny mnou zkoumané nástroje na zpracování přirozeného jazyka nabízejí určitou možnost pro využití v českém jazyce. Většina nástrojů umožňuje převod textu do unicode kódování. Po zpracování je možnost převést text zpět do původního kódování.

Během práce jsem získal obecný přehled o principech zpracování přirozeného jazyka, čehož si velice vážím. Jak jsem již v úvodu zmínil, tak zpracování přirozeného jazyka má veliký potenciál a zasahuje do různých odvětví. Proto doufám, že znalosti nabyté během práce na bakalářském projektu mi pomohou v budoucí praxi.

Literatura

[1] Čermák F. Jazyk a jazykověda, pages 126-131. Karolinum, Praha, 2004. ISBN 80-246-0154-0.

[2] NOVOTNÝ, Jiří a kolektiv. Mluvnice češtiny pro střední školy. Fortuna, Praha, 1992. ISBN 80-85298-32-5.

[3] Natural Language Toolkit. Dostupné na URL http://www.nltk.org/Home (duben 2009)

[4] Python Software Foundation. Python v2.6.2 documentation. Dostupné na URL http://docs.python.org/ (duben 2009).

[5] Černý Stanislav, Bc. Morfologický analyzátor pomocí konečných automatů. FIT VUT v Brně, 2008. Bakalářská práce.

[6] Wikipedia, otevřená encyklopedie. Interpunkční znaménko. Dostupné na URL http://cs.wikipedia.org/wiki/Interpunkční_znaménko (duben 2009).

[7] Karlík, P.; Nekula, M.; Rusínová, Z. Příruční mluvnice češtiny. Nakladatelství Lidové Noviny, Praha, 1995. ISBN 80-7106-134-4.

[8] Steven Bird, Ewan Klein, Edward Loper. Natural Language Processing with Python. O´Reilly, Sebastopol, CA, 2009. ISBN 10: 0-596-51649-5. Dostupné na URL http://www.nltk.org/book (duben 2009).

[9] Alias-i, Inc. LingPipe. Dostupné na URL http://alias-i.com/lingpipe/ (duben 2009).

[10] OpenNLP. Dostupné na URL http://opennlp.sourceforge.net/ (duben 2009).

[11] Edward Loper and Steven Bird. *Proceedings of the ACL Workshop on Effective Tools and Methologies for Teaching Natural Language Processing and Computational Linguisticks*. pp 62-69, Philadelphia, Association for Computational Linguistics. July 2002. Dostupné na URL http://arxiv.org/PS_cache/cs/pdf/0205/0205028v1.pdf(duben 2009)

Seznam příloh

Příloha 1. Uživatelský manuál

Příloha 2. CD se zdrojovými texty a tímto obsahem:

- Adresář */czech* - zdrojové texty
- PDF soubor *technická_zpava.pdf* – úplná technická zpráva ve formátu PDF
- PDF soubor *rules.pdf* – princip tvoření pravidel ve skriptu *chunky.py*

Příloha 1. Uživatelský manuál

Před samotným užitím jednotlivých skriptů je nutné obsah adresáře *czech* nakopírovat do adresáře k zdrojovému textu, který bude skripty užívat. Jednotlivé skripty využívají knihovnu *libma* kolegy Bc. Černého [5]. Je tedy nutné do adresáře, kde jsou umístěny skripty přikopírovat rozhraní této knihovny pro jazyk Python.

Skript cip.py

inflect(word,number,case)

Parametry:

word = slovo, které se zpracovává. [parametr typu string]
number = mluvnické číslo. Př.: „S“ - jednotné číslo, „P“ - množné číslo.[parametr typu string]
case = pád. Př.: 1 - 7 [parametr typu int]

Výstupem je soubor řešení, který má tvar: ['řešení1','řešení2','řešení3',atd.]

inflex(word,person,number,time)

Parametry:

word = slovo, které se zpracovává. [parametr typu string]
person = osoba. Př.: 1 – 3 [parametr typu int]
number = mluvnické číslo. Př.: „S“ - jednotné číslo, „P“ - množné číslo.[parametr typu string]
time = čas. Př.: „past“ =čas minulý, „present“=čas přítomný, „future“=čas budoucí.
[parametr typu string]

Výstupem je soubor řešení, který má tvar: ['řešení1','řešení2','řešení3',atd.]

lemma(word,number)

Parametry:

word = slovo, které se zpracovává. [parametr typu string]
number = počet řešení zobrazených na jeden řádek. [parametr typu int]

Výsledek je vypsán na standardní výstup.

inflection (words,case)

Parametry:

word = slovní spojení. [parametr typu string]
case = pád. Př.: 1 - 7 [parametr typu int]

inflexion (words)

Parametry:

word = slovní spojení. [parametr typu string]

Výsledek je soubor všech variant.

Skript split.py

Split(file,input)

Parametry:

file = název souboru nebo textový řetězec. [parametr typu string]

input = příznak pro rozlišení hodnoty parametru *file*. [parametr typu string]

Př.: „f“ = vstup je soubor(název souboru)

„io“ = vstup je textový řetězec.

Při vstupu souboru je výsledkem soubor s označenými částmi textu. Pokud je vstupem textový řetězec, výstupem je označený řetězec.

PoSplit(file,input)

Parametry:

file = název souboru nebo textový řetězec. [parametr typu string]

input = příznak pro rozlišení hodnoty parametru *file*. [parametr typu string]

Př.: „f“ = vstup je soubor(název souboru)

„io“ = vstup je textový řetězec.

Výstupem je soubor jednotlivých vět. Př.: ['věta1','věta2','věta3',atd.]

Skript chunky.py

Chunk(sentence, rules,flagy)

Parametry:

sentence = věta s označenými slovy. [parametr typu string]

rules = soubor transformačních pravidel.[parametr typu string]

Př.: ['pravidlo 1','pravidlo 2',atd.]

flagy = příznak pro rozlišené formy zobrazení výsledku. [parametr typu string]

Př.: „NOR“ = standardní výstup.

„API“ = grafické zobrazení.